

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-227405

(43)Date of publication of application : 03.09.1996

(51)Int.Cl.

G06F 15/16

G06F 17/12

G06F 17/16

(21)Application number : 07-032099

(71)Applicant : HITACHI LTD

(22)Date of filing : 21.02.1995

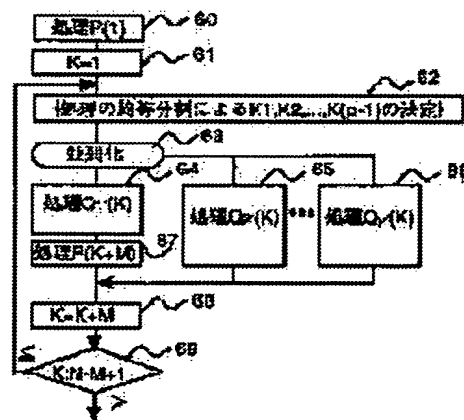
(72)Inventor : TANAKA TERUO
YAMAMOTO YUSAKU
HOJO YOSHIKI
TAMAOKI YOSHIKO
SAKAKIBARA TADAYUKI
ASAYA MACHIKO
YASUDA YOSHIKO

(54) PARALLEL EXECUTING METHOD FOR REPETITIVE PROCESSING

(57)Abstract:

PURPOSE: To execute a repetitive processing such as an LU decomposing processing by parallel computers in a shorten time.

CONSTITUTION: For example, a (K)th repetitive processing consists of a processing P(K) which can not be performed in parallel and a processing Q(K) which can be performed in parallel, the processing Q(K) uses the result of the processing P(K), and a (K+1)th the processing P(K+1) uses the result of a processing QS(K) as part of the (K)th processing Q(K). In this case, the processing Q(K) is decomposed into processings Q1(K)-QP(K) as many as the number P of processors, and the processing QS(K) is included in the processing Q1(K). The processing Q1(K) and processing P(K+M) are executed in order by the same processor and other processing Q2(K)-Qn(K) are executed by other (P-1) processors in parallel. The said processings are repeated thereafter each time of processes of all the processors end.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision
of rejection]

[Date of requesting appeal against examiner's
decision of rejection]

[Date of extinction of right]

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-227405

(43) 公開日 平成8年(1996)9月3日

(51) Int. Cl. ⁶	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 15/16	3 9 0		G 0 6 F 15/16	3 9 0 Z
17/12			15/324	
17/16			15/347	K

審査請求 未請求 請求項の数14 O L (全 16 頁)

(21) 出願番号 特願平7-32099

(22) 出願日 平成7年(1995)2月21日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 田中 輝雄

東京都国分寺市東恋ヶ窪1丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 山本 有作

東京都国分寺市東恋ヶ窪1丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 北城 芳章

東京都江東区新砂一丁目6番27号 株式会

社日立製作所公共情報事業部内

(74) 代理人 弁理士 蔭田 利幸

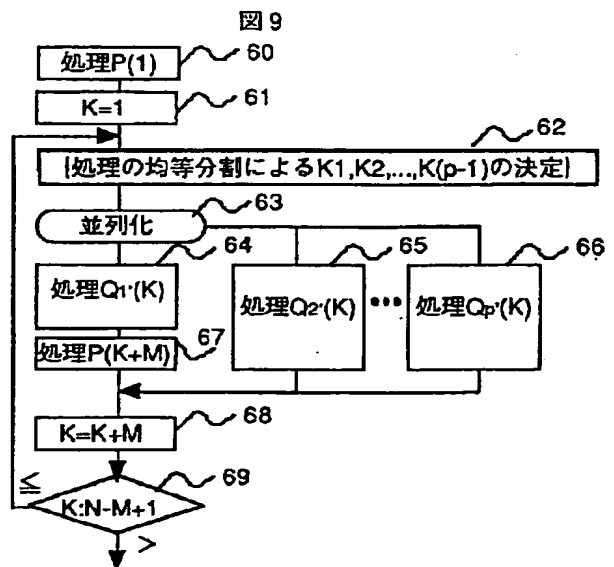
最終頁に続く

(54) 【発明の名称】 反復処理の並列実行方法

(57) 【要約】

【目的】 LU分解処理などの反復処理を並列計算機により短時間に実行可能にする。

【構成】 K回目の反復処理が、並列化不可能な処理P(K)と並列化可能な処理Q(K)とから構成され、処理Q(K)は、処理P(K)の結果を利用し、かつ、K+1回目の処理P(K+1)はK回目の処理Q1(K)中の一部の処理Qs(K)の結果を利用する時、処理Q(K)を、プロセッサの台数Pと同数の処理Q1(K)からQP(K)に分解し、かつ、処理Q1(K)には、処理Qs(K)を含ませる。処理Q1(K)と処理P(K+M)を同一のプロセッサで順次実行し、これと並行して、他の処理Q2(K)からQn(K)を他のP-1台のプロセッサで実行する。以後、全プロセッサの処理が終了する毎に以上の処理を繰り返す。



【特許請求の範囲】

【請求項 1】第 1 の処理と、その実行結果を使用する第 2 の処理とを反復して実行する反復処理であって、第 2 の処理は、並列に実行可能な複数の部分処理からなり、第 1 の処理は、該反復処理のある繰返し時における第 2 の処理の実行結果をその反復処理の次の繰返し時に使用するものを並列計算機で実行させる方法であって、

(a) 該並列計算機を構成する複数のプロセッサのいずれか一つにより、該反復処理の第 1 の繰返しに対して第 1 の処理を実行させ、

(b) 一つの処理群が、第 2 の処理に含まれる上記複数の部分処理の内、第 1 の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第 1 の処理とを少なくとも含むように、かつ、他の処理群が、第 2 の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、該第 1 の処理と第 2 の処理に含まれた上記複数の部分処理とを、上記複数のプロセッサの数に等しい数の複数の処理群に分割し、

(c) 該第 1 の繰返しに対して、該ステップ (a) の実行結果を使用して該一部の複数の部分処理を実行し、該次の繰返しに対して、該一部の複数の部分処理のこの実行結果を使用して第 1 の処理を実行するように、該一部の複数の部分処理と該第 1 の処理を少なくとも含む上記一つの処理群を、該複数のプロセッサのいずれか一つにより実行し、

(d) 該他の複数のプロセッサの各々により、該複数の処理群の一つを、該第 1 の繰返しに対して該ステップ

(a) の実行結果を使用して、かつ、該一つのプロセッサによるステップ (c) での上記実行と並行して実行し、

(e) 各プロセッサによる、ステップ (c) または (d) の実行の終了後、上記ステップ (c) から (e) を繰返し、その繰返しにおいては、(e 1) 該第 2 の処理を該次の繰返し以降の繰返しに対して実行され、かつ、該第 1 の処理を該次の繰返し以降の繰返しの次の繰返しに対して実行されるように、ステップ (c) から (d) を実行し、(e 2) 上記繰返しをする前にステップ (c) の実行の結果得られた、第 1 の処理の実行結果を、上記繰返しをした後に実行されるステップ (d) において第 2 の処理により利用する反復処理の並列実行方法。

【請求項 2】ステップ (e) による繰返しのときに、ステップ (b) も実行した上で繰り返す請求項 1 記載の反復処理の並列実行方法。

【請求項 3】ステップ (b) は、該一部の複数の部分処理に付加して該一つのプロセッサにより実行すべき他の一部の複数の部分処理が、該一つの処理群に属するように、該分割を実行するステップを有する請求項 1 記載の反復処理の並列実行方法。

【請求項 4】ステップ (b) は、上記一部の複数の部分処理と上記第 1 の処理の実行に必要な第 1 の処理時間と、該他の複数の部分処理の各々の実行に必要な第 2 の処理時間とに基づいて、該複数の処理群の実行時間がほぼ等しくなるように、上記分割を行なうステップを有する請求項 1 記載の反復処理の並列実行方法。

【請求項 5】上記反復処理は、N 次元連立一次方程式の係数行列を LU 分解する処理であり、上記第 1 の処理はピボット処理を含み、上記第 2 の処理は係数行列の消去更新処理を含む請求項 1 から 4 のいずれか一つに記載の反復処理の並列実行方法。

【請求項 6】ステップ (e) は、各プロセッサにおける処理の終了を検出した後、該一つのプロセッサによるステップ (c) の繰返しおよび該他の複数のプロセッサの各々によるステップ (d) の繰返しを同時に起動するステップを有する請求項 1 記載の反復処理の並列実行方法。

【請求項 7】ステップ (e) は、各プロセッサにより、そのプロセッサでの処理の終了後に、他の複数のプロセッサの各々での処理の終了を判別し、各プロセッサにより、他のプロセッサの各々での処理の終了が検出されたときに、そのプロセッサでの、ステップ (c) または (d) の繰返しを起動するステップを有する請求項 1 記載の反復処理の並列実行方法。

【請求項 8】該並列計算機は該複数のプロセッサにより共有された主記憶をさらに有し、該方法は、

(f) 各プロセッサが該ステップ (c) または (d) の実行を終了したとき、その終了を示す情報をそのプロセッサにより該主記憶に書き込むステップをさらに有し、該ステップ (e) は、各プロセッサから該主記憶に書き込まれた、そのプロセッサでの処理の終了を示す情報に基づいて行なわれる請求項 7 記載の反復処理の並列実行方法。

【請求項 9】ステップ (f) は、ステップ (c) の実行の終了後、該一つのプロセッサによりそのプロセッサでの処理の終了を示す情報を該主記憶に書き込み、該他の複数のプロセッサの各々により、ステップ (d) による、いずれかの部分処理群の実行の終了後に、そのプロセッサでの処理の終了を示す情報をその各プロセッサにより該主記憶に書き込むステップを有し、ステップ (e) は、

該一つのプロセッサにより、上記終了を示す情報を書き込んだ後に、該他の複数のプロセッサの各々による、終了を示す情報が該主記憶に書き込まれたか否かを該一つのプロセッサにより監視し、該他の複数のプロセッサの各々による、上記終了を示す情報の書き込みの後に、該他の複数のプロセッサの以外

のプロセッサの各々による、終了を示す情報が該主記憶に書き込まれたか否かを、該他の複数のプロセッサの各々により監視することにより、該複数のプロセッサの各々により、他のプロセッサでのステップ(c)または(d)での処理の実行が終了したか否かを判別するステップとを有し、

ステップ(e)は、該ステップ(e)による各プロセッサ別の、他プロセッサの終了の判別結果に基づいて、各プロセッサ毎に、ステップ(c)から(d)の繰返しの開始時期を決定するステップを有する請求項8記載の反復処理の並列実行方法。

【請求項10】第1の処理と、その実行結果を使用する第2の処理とを反復して実行する反復処理であって、第2の処理は、並列に実行可能な複数の部分処理からなり、第1の処理は、該反復処理のある繰返し時における第2の処理の実行結果をその反復処理の次の繰返し時に使用するものを並列計算機で実行させる方法であって、

(a) 該並列計算機を構成する複数のプロセッサのいずれか一つにより、該反復処理の第1の繰返しに対して第1の処理を実行させ、

(b) 一つの処理群が、第2の処理に含まれる上記複数の部分処理の内、第1の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第1の処理とを少なくとも含むように、かつ、他の処理群が、第2の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、第2の処理に含まれた上記複数の部分処理と第1の処理とを上記複数のプロセッサの数に等しい数の複数の処理群に分割し、

(c) 該第1の繰返しに対して、該ステップ(a)の実行結果を使用して該一部の複数の部分処理を実行し、該次の繰返しに対して、該一部の複数の部分処理のこの実行結果を使用して第1の処理を実行するように、該一部の複数の部分処理と該第1の処理を少なくとも含む上記一つの処理群を、該複数のプロセッサのいずれか一つにより実行し、

(d) 該他の複数のプロセッサの各々により、該複数の処理群の一つを、該第1の繰返しに対して、該ステップ(a)の実行結果を使用して、かつ、該一つのプロセッサによるステップ(c)での上記実行と並行して実行し、

(e) 各プロセッサにおいて、そのプロセッサによるステップ(c)または(d)の実行が終了した時点で、そのプロセッサにより上記(c)または(d)を繰返し、その繰返しにおいては、(e1)ステップ(c)または(d)で実行した処理群とは異なる他の一つの処理群をそのプロセッサで実行するように、ステップ(c)または(d)を実行し、(e2) 当該他の処理群が第2の処理の部分処理群を含んでいるときには、それらの部分処理群を該次の繰返し以降の繰返しに対して実行し、当該他

の処理群が第1の処理を含んでいるときには、その第1の処理を該次の繰返し以降の繰返しの次の繰返しに対して実行し、(e3) そのプロセッサがステップ(c)または(d)で実行した処理群が第1の処理を含んでいないときには、他のプロセッサから通知される、第1の処理の実行結果の通知を確認した後に、上記繰返しを開始するステップを有する反復処理の並列実行方法。

【請求項11】ステップ(e)による該繰返しは、各プロセッサがステップ(e)による繰返し毎に、順次異なる処理群を実行するように行なわれる請求項10記載の反復処理の並列実行方法。

【請求項12】いずれか一つのプロセッサによりステップ(c)によりいずれかの繰返しに対して上記一つの処理群を実行したときに、その実行結果を他の複数のプロセッサに通知し、

他の複数のプロセッサの各々は、上記実行結果が、該一つのプロセッサから通知されたのを確認してから、その繰返しに対する、該複数の処理群の内のいずれかの他の一つの処理群を実行し、

該一つの処理群を上記繰返しの次の繰返しに対して実行する、さらに他のプロセッサに、該他の複数のプロセッサの各々により該他の処理群の実行結果を通知し、該さらに他のプロセッサでは、そのプロセッサ以外の複数のプロセッサから上記繰返しに対する上記複数の処理群の実行結果の受信を、当該さらに他のプロセッサにより確認してから、上記さらに次の繰返しに対して、該一つの処理を開始するステップをさらに有する請求項10記載の反復処理の並列実行方法。

【請求項13】 該並列計算機は、該複数のプロセッサを相互に接続するネットワークと、各プロセッサに対応して設けられ、そのプロセッサで実行されるプログラムとデータを保持するローカルメモリとを有する分散記憶型並列計算機であり、

上記一つの処理の実行結果の通知と上記他の処理群の実行結果の通知は、このネットワークを介して行なわれる請求項12記載の反復処理の並列実行方法。

【請求項14】 第1の処理と、その実行結果を使用する第2の処理とを反復して実行する反復処理であって、第2の処理は、並列に実行可能な複数の部分処理からなり、第1の処理は、該反復処理のある繰返し時における第2の処理の実行結果をその反復処理の次の繰返し時に使用するものを、それぞれ複数のプロセッサとそれらのプロセッサで共有される、プログラムとデータを保持するためのローカルメモリとを有する複数のクラスタと、それらを結合するネットワークからなる分散記憶型の並列計算機で実行させる方法であって、

(a) 該複数のクラスタ内のいずれか一つに含まれた複数のプロセッサのいずれか一つにより、該反復処理の第1の繰返しに対して第1の処理を実行させ、

(b) 一つの処理群が、第2の処理に含まれる上記複数

の部分処理の内、第1の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第1の処理とを少なくとも含むように、かつ、他の処理群が、第2の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、該第1の処理と第2の処理に含まれた上記複数の複数の部分処理とを、上記複数のクラスタの数に等しい数の複数の処理群に分割し、

(c) 該複数のクラスタの各々により、該複数の処理群の一つを他のクラスタによる他の処理群の実行と並行して実行し、各クラスタによる、一つの処理群の実行に当たっては、(c1) その一つの処理群に上記一部の複数の部分処理と該第1の処理とが含まれているときには、その一部の複数の部分処理と第1の処理とを少なくとも含む一つの部分処理群と、第2の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部をそれぞれ含む複数の他の部分処理群とに、該一つの処理群の処理を分割し、該一つの処理群に上記一部の複数の部分処理と第1の処理が含まれていないときには、第2の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部をそれぞれ含む複数の他の部分処理群に、該一該一つの処理群の処理を分割し、(c2) そのクラスタ内の複数のプロセッサにより、ステップ(c1)による分割で得られた複数の部分処理群の一つを、そのクラスタ内の他のプロセッサによる他の部分処理群の実行と並行して実行し、(c3) そのクラスタで実行した上記一つの処理群に上記第1の処理が含まれている場合、その第1の処理が属する一つの部分処理群を実行した一つのプロセッサから、他のクラスタに第1の処理の実行結果を通知し、

(d) 各クラスタにおいて、そのクラスタ内の各プロセッサによるステップ(c1)の実行が終了した時点で上記(c)を繰返し、その繰返しにおいては、(d1) 該複数の処理群のうち、そのクラスタによりステップ

(c)で実行した処理群とは異なる他の処理群をそのクラスタで実行するように、ステップ(c)を繰返し、

(d2) ステップ(c)でそのクラスタが実行した処理群の中に、該第1の処理が含まれていない場合には、他のクラスタから通知されるその第1の処理の実行結果の通知を確認した後、該他の処理群の実行を開始し、(d3) その繰返しにおいて実行する該他の処理群が、第2の処理に属する部分処理群を含んでいる場合には、該次の繰返し以降の繰返しに対してその部分処理群を実行し、かつ、該他の処理群が、第1の処理を含んでいるときには、該次の繰返し以降の繰返しの次の繰返しに対して第1の処理を実行するように、該他の処理群を実行する反復処理の並列実行方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、N次元連立一次方程式の係数行列を下三角行列と上三角行列に分解するLU分解処理のように、同じ処理を異なるデータに対して反復して実行する反復処理を、複数のプロセッサを持つ並列計算機で実行させる方法に関する。

【0002】

【従来の技術】各種の工業用製品の開発に数値シミュレーションが実用に供されている。このような数値シミュレーションでよく使用される処理は、N次元連立方程式 $AX=B$ をガウスの消去法で解く処理である。ここで、AはN行N列の係数行列、XとBはN列のベクトルである。このガウスの消去法により上記方程式を解く処理の主要な部分は、係数行列Aを下三角行列Lと上三角行列Uの積である行列LUに分解するLU分解処理である。

【0003】大規模行列のLU分解処理の実用化は進んでおり、特にスーパーコンピュータ向けには、多段同時展開法が一般的である。これらの手法は、文献：村田、小国、唐木著スーパーコンピュータ、丸善、91~103頁、1985や文献：ブロッキングLU分解法のVP2000シリーズ向けチューニングについて、情報処理学会第42回(平成3年前期)全国大会予稿集第1分冊71~72頁、特開平4-259064および特開平6-75988などに開示されている。

【0004】とくに最近は、並列計算機の進歩に伴い、並列計算機を数値シミュレーションに応用する研究が盛んである。

【0005】近年実用化されている並列計算機は、一般的に次のような3種類の形態をとる。図1は、複数のプロセッサ2ないし4が主記憶1を共有する主記憶共有型並列計算機である。これは、現在のベクトル型スーパーコンピュータの主流であり、スーパーコンピュータの場合、各プロセッサ2ないし4は多数のデータをベクトルデータとしてまとめ、そのベクトルデータに同一の処理をパイプライン演算器を使用して連続的に施すベクトルプロセッサとなる。

【0006】図2は、各々が演算処理装置10と専用の主記憶9を持つ独立したを計算機6ないし8である。ここで、計算機6ないし8をプロセッサと呼ぶ。これらのプロセッサ6ないし8を相互結合網5で接続する。現在は、各プロセッサはスーバスカラなどの高性能マイクロプロセッサとメモリで構成されることが多い。

【0007】図3は、主記憶15および複数のプロセッサ16ないし17から構成される主記憶共有型並列計算機12ないし14(図1と同様)をさらに相互結合網11で接続した構成を持つ。これは、たとえば、特開平6-96032で開示されている。図3の構成における各々の主記憶共有型計算機12ないし14をクラスタと呼ぶ。

【0008】図2の形式の並列計算機でLU分解処理を実行させる方法の例は、文献：星野著PAXコンピュー

タ、オーム社、1985や特開平3-105694などに開示されている。

【0009】LU分解処理の概要および図1に示した並列計算機でLU分解処理を実行させる従来の方法を図4、図5を用いて説明する。

【0010】図4は、行列Aに対するLU分解処理のK回目のループ処理を説明するための図である。すなわち、行列Aの各要素 $a(i, j)$ （すなわち、第i行第j列番目の要素）を、異なる処理を受ける要素群に分けて、それらの要素群を領域の形で示している。図中、Nは行列サイズである。ここでは、第1行から第K-1行および第1列から第K-1列までのLU分解処理は終了している。各反復回では、高速化のために、一度にM列をまとめてLU分解処理を行なういわゆる多段同時消去方法が使用される。このためには、Mは複数であるが、原理的にはMは1でもよい。

【0011】以下の説明のため、行列Aの第K列から第N列ないし第K行から第N行の領域を5つに分け、それぞれに次のような名称を付ける。

【0012】(1) L1(K) 22: 第K列から第K+M-1列および第K行から第K+M-1行で構成される左下三角領域

(2) U1(K) 21: 第K列から第K+M-1列および第K行から第K+M-1行で構成される右上三角領域

(3) L2(K) 23: 第K列から第K+M-1列および第K+M行から第N行で構成される矩形領域

(4) U2(K) 24: 第K+M列から第N列および第K行から第K+M-1行で構成される矩形領域

(5) A(K) 25: 第K+M列から第N列および第K+M行から第N行で構成される矩形領域

図5に、図1の並列計算機で実行可能な、従来技術によるLU分解処理プログラムを示す。この図において、LU分解プログラムは、2重のDOループで構成されている。

【0013】外側のループ20は、M列づつまとめて処理するために、行列のサイズNをMで割ったM/N回だけ繰り返す。

【0014】内側は2つDOループで構成され、それらの処理は以下のとおりである。なお、外側ループ20の繰返し回数はKとする。

【0015】まず、最初の内側ループ(DO 1)の処理では、第K列から第K+M-1列番目まで、つまりU1(K) 21、L1(K) 22およびL2(K) 23の領域に対して、各列ごとに、部分軸選択（いわゆるピボット処理）およびそれに対する消去更新処理を繰り返して行なう。これらの処理の概要は以下のとおりである。最初の内側ループ(DO 1)のループ繰返し回数はKMとする。但し、KMはK、Mとは関係のない値である。

【0016】(1) ピボット処理: KM列の第KM行か

ら第N行までの要素から最大値を検索し、検索した最大値を含む行と第KM行との行交換を行なう。すなわち、それらの行の第KM列から第K+M-1列番目の要素を交換する。

【0017】(2) 消去更新処理: 第KM+1列から第K+M-1列および第KM+1行から第N行により構成される矩形領域内の各要素 $a(i, j)$ に対して、消去更新処理 $a(i, j) = a(i, j) - a(KM, j) \times a(i, KM)$ を行なう。

【0018】このDOループの処理を、以下では、処理P(K)と呼ぶ。このDOループの処理では、最大値検索という逐次処理を繰り返して行なう必要があるため、このループ自体を並列化してもあまり性能が向上しない。

【0019】次に、第2の内側DOループ(DO 2)では、領域U2(K)および領域A(K)に対して、次の処理を行なう。

【0020】(1) 領域U2(K)および領域A(K)の行交換: 前述のピボット処理において行交換処理を施したM組の行の第K+M列から第N列の要素に対して、行交換処理を行なう。

【0021】(2) 領域U1(K)、領域L1(K)を用いた領域U2(K)の消去更新処理: この領域U2(K)内の各要素 $a(i, j)$ に対して、消去更新処理 $a(i, j) = a(i, j) - a(k, j) \times a(i, k)$ を行なう。ここで、kをKからK+M-1に順次変化させて、この計算を合計M回行なう。なお、iは、K+1からK+M-1の一つを1取り、jはK+MからNの一つの値を取る。したがって、この計算は、領域U1(K)、L1(K)内の要素を使用して、領域U2(K)内の要素を更新している。

【0022】(3) 領域U2(K)、L2(K)を用いた領域A(K)内の要素の消去更新処理: 消去更新処理 $a(i, j) = a(i, j) - a(k, j) \times a(i, k)$ を行なう。ここで、kをKからK+Mに順次変えて、この計算を合計M回行なう。ここで、iはk+MからNの一つであり、jはK+MからNの一つである。

【0023】この第2の内側ループの処理を、以下では、処理Q(K)と呼ぶ。

【0024】これらの計算(1)から(3)は、領域U2(K)、A(K)の各列ごとに独立に計算可能である。すなわち、各列毎に、上記処理(1)から(3)を実行する処理が基本の処理であり、処理Q(k)は、この単位処理を複数の列に対して集めたものとなっている。このため、このような各列毎の処理は、処理Q(K)を構成する、互いに並列に実行可能な部分処理と言える。したがって、Q(K) 28を列を単位に均等にp個にグループに分け、それぞれをQ1(K)、Q2(K)、...、QP(K)とし、p台のプロセッサに分割して処理を行なうことにより、p倍に近い台数効果を得

ることができる。

【0025】このようなLU分解処理のフローチャートは、図7の通りである。図7では、パラメタKの初期値設定35に続き、処理P(K)36の実行後、並列化可能な処理Q(K)の並列化37を行ない、p個の処理Q1(K)38ないし処理Qp(K)40に分け、p台のプロセッサでそれぞれの処理を分担し、並列に実行する。このとき、並列化された処理Q(K)のあとに、処理を逐次にもどして、Kの更新処理41と収束判定42を行ない、次のK+M回目の反復ループである処理P(K+M)、さらに処理Q(K+M)を実行する。

【0026】このように処理Q(K)を複数の並列実行可能な処理に分けて実行するプログラムを得るには、最適化コンパイラの自動並列化機構を使用してソースプログラムの内の並列実行可能な部分を検出し、かつ、それらを自動的に並列実行可能なオブジェクトプログラムに変換する方法あるいはソースプログラムに並列指示行を追加し、これをコンパイラにより並列実行可能なオブジェクトプログラムに変換する方法が取られる。図5に示したプログラムは、DO ALLタイプの並列化指示行29を有する例である。このプログラムの場合、処理Q(K)28の並列化は並列指示行29により各反復回ごとに毎行なわれる。

【0027】以上の説明から明かな通り、LU分解処理の基本的な処理の流れは、図6に示すように、パラメタKの初期値設定30およびそのパラメタの更新処理33により制御し、処理P(K)31と、処理Q(K)32を指定した条件34(収束条件あるいは反復回数など)が成立するまで繰返し実行することである。図5のプログラムは、この処理を、図7に示す手順にしたがって図1の形式の並列計算機で実行可能にし、それでもってLU分解処理の高速化を図っている。

【0028】数値シミュレーションでは、LU分解処理に限らず、ほぼ同一の処理を、図6に示す手順にしたがって繰返す反復処理が多く見受けられる。その場合は、Mは1でもよい。このような他の反復処理でも、処理Pが本来逐次的にしか実行できないとしても、図7に示す処理手順にしたがって処理Qを並列に実行することにより、このような他の反復処理も高速に実行できる。

【0029】

【発明が解決しようとする課題】LU分解処理を図7に示した手順で処理した場合、処理Q(K)が並列計算機を構成する複数のプロセッサにより並列に実行されるため、その処理の実行時間は、プロセッサ数が多いほど短縮される。しかし、各反復回ごとに処理P(K)は並列計算機を構成する一つのプロセッサにより実施されるため、この処理P(K)の実行時間が、プロセッサ数を多くしても、LU分解処理全体の処理時間を短縮するのを妨げている。

【0030】いま仮に、対象とする行列のサイズNを1

000としたとき、上記処理Q(K)と処理P(K)は、それぞれLU分解処理の総演算量の処理のおおよそ98%および2%程度の演算量を占める。しかし、処理P(K)、処理Q(K)の実行時の演算器の使用率が大幅に異なる。

【0031】たとえば、図1の並列計算機が4台のプロセッサからなり、各プロセッサがベクトルプロセッサの場合、この処理Q(K)は並列処理可能であり、この処理のために異なるベクトルプロセッサ内のパイプライン演算器も使用される上に、処理Q(K)の計算処理は積和計算のため、各ベクトルプロセッサ内のパイプライン演算器の使用率が高い。一方、処理P(K)は、一台のベクトルプロセッサ内の演算器しか使用しない上に、この処理の主たる部分は最大値検索処理であり、必要とするデータ量に対して相対的に演算量が少なく、この一つのベクトルプロセッサ内の一部のパイプライン演算器しか使用せず、演算器の使用率は高くない。このため、上述の4台のベクトルプロセッサからなる並列計算機に対する推定例では、処理Q(K)に対する処理P(K)の演算器利用率は約1/10程度と低い。このため、処理P(K)と処理Q(K)の処理時間の比は、

$$\text{約 } 10 : 12 (= 2 : 98 / (10 \times 4))$$

となり、わずか2%の演算量の処理P(K)がLU分解処理全体の処理時間の約半分を占める。したがって、4台のプロセッサを使用しているにもかかわらず最大2台分の性能向上しか得られない。

【0032】また、図1のプロセッサ1等が、スーパースカラプロセッサからなり、かつプロセッサの総数が100台程度である場合の推定例では、処理Q(K)に対する処理P(K)の演算器利用率は約1/2である。このため、処理P(K)と処理Q(K)の処理時間の比は、

$$\text{約 } 4 : 1 (= 2 : 98 / (2 \times 100))$$

となり、わずか2%の演算量の処理P(K)が演算時間の約80%を占めることがわかる。したがって、100台のプロセッサを使用しているにもかかわらず10台強の性能向上しか得られない。

【0033】したがって、処理P(K)の実行時間が、LU分解処理の全体の処理時間に及ぼす影響をさらに減じることが望まれる。

【0034】同じような問題は、LU分解処理に限らず、処理P(K)で例示される第1の処理と、その処理の実行後に実行すべき、処理Q(K)で例示される並列実行可能な第2の処理とを繰返するとともに、この繰返しに当たっては、第2の処理の実行後に第1の処理を実行するように第1、第2のの処理を実行するときに生じる。

【0035】言い替えると、従来の方法では、複数のプロセッサの内、一部のプロセッサは処理をしないで、他のプロセッサでの処理待ちの状態にある時間が長いこと

になる。

【0036】したがって、本発明の目的は、本来は逐次処理されるべき第1の処理とその後実行されるべき、並列実行可能な第2の処理とを有し、第2の処理の完了を待って再度第1、第2の処理を実行するように、第1、第2の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、この反復処理全体の処理時間を軽減することが可能な反復処理の並列実行方法を提供することである。

【0037】本発明の他の目的は、このような処理の実行時に、プロセッサが待ち状態にある時間を軽減することである。

【0038】本発明のより具体的な目的は、LU分解処理を並列計算機で実行させる場合に、この処理全体の処理時間を軽減することが可能なLU分解処理の並列実行方法を提供することである。

【0039】本発明の他のより具体的な目的は、このLU分解処理を実行しているときのプロセッサが待ち状態にある時間を減少させることである。

【0040】

【課題を解決するための手段】上に説明したLU分解処理では、外側ループの繰返し回数Kのときに、処理P(K)の結果を使用して実行される処理Q(K)では、領域U2(K)と領域A(K)の要素の更新後の値が計算される。これらの要素のうち、外側ループの次の繰返し時に実行される処理P(K+M)の計算に使用する要素は、このK+1回目における領域U1(K)、L1

(K)、L2(K)に属すべき要素のみである。すなわち、図4において、第K+M行から第N行と第K+M列から第K+2*M-1列により構成される領域26内の要素である。したがって、処理P(K+M)は、この領域26内の要素に関する処理Q(K)（これをQS

(K)と呼ぶことにする）が終了後に実行を開始しなければならない。しかし、処理P(K+M)は、領域A(K)の内、この領域26以外の領域内の要素に対する処理Q(K)とは、並列に実行可能である。

【0041】本発明では、このことを利用して、本来は逐次処理されるべき第1の処理（例えば、P(K)、以下括弧内は例示である）とその後実行されるべき、並列実行可能な第2の処理（Q(K)）とを有し、第2の処理の完了を待って再度第1、第2の処理を実行するように、第1、第2の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、ある繰返し(K回目)で実行される第2の処理（Q(K)）の内、次の繰返し(K+M回目)で実行されるべき第1の処理（P(K+M)）が使用する処理結果を生成する部分処理（QS(K)）と、次の繰返し時の第1の処理（P(K+M)）とを同じプロセッサに割り当て、この第1の処理（P(K+M)）を、この部分処理（QS(K)）に続いてそのプロセッサで実行させる。第2の処理（Q

(K)）の内、この部分処理（QS(K)）以外の部分を、残りのプロセッサの数P-1に等しい数の、並列に実行可能な処理（Q2(K)からQP(K)）に分割し、それらの残りのプロセッサにより並列に実行させる。

【0042】この際、上記部分処理（QS(K)）と第1の処理（P(K+M)）とを実行するプロセッサの処理時間が、他のプロセッサのそれより小さいときには、前者の処理時間が他のプロセッサのそれとほぼ同じになるように、第2の処理（Q(K)）の内、部分処理（QS(K)）以外の一部もこの部分処理（QS(K)）を割り当てたプロセッサに割り当てることがより望ましい。すなわち、第2の処理（Q(K)）を実施するのに必要な第1の処理（P(K)）を、いずれかのプロセッサにより予め実行した後、上記のごとくにして各プロセッサに割り当てべき第2の処理（Q(K)）の一部を決定し、決定された処理の割り当てに従って、各プロセッサで、部分処理（QS(K)）もしくはQi(K)（i=2からp-1））を実行させる。

【0043】部分処理（QS(K)）を割り当てられたプロセッサでは、この処理が終了すると、第1の処理（P(K+M)）を実行させる。この部分処理（QS(K)）と第1の処理（P(K+M)）および他の部分処理（Q2(K)からQP(K)）の実行の終了を待って、以上の処理を繰り返す。

【0044】なお、本発明は、図2、図3の形式の並列計算機でも実施可能である。

【0045】

【作用】次の反復回（(K+M)）の第1の処理（P(K+M)）は、第2の処理（Q(K)）の内、その第1の処理（P(K+M)）に先立って実行する必要がある部分処理（QS(K)）以外の処理と並列に実行されるので、並列計算機内のプロセッサが待ち状態にある時間を軽減することが出来、計算機資源の有効な利用を図ることが出来る。それにより、この反復処理全体の処理時間を軽減することが可能になる。又、従来のごとく、第1の処理（P(K+M)）を第2の処理（Q(K)）と逐次に行う従来の場合に比べて、この第1の処理（P(K+M)）の実行時間が、繰返し処理全体に及ぼす影響を減少できる。なお、第1の処理（P(K+M)）は、部分処理（QS(K)）を割り当てられたプロセッサ内で、この処理の後に実行されるので、正常な結果を出力する。

【0046】

【実施例】以下、本発明に係る反復処理の並列実行方法を図面に示したいくつかの実施例を参照してさらに詳細に説明する。なお、以下においては、同じ参照番号は同じものもしくは類似のものを表わすものとする。

【0047】＜実施例1＞本実施例では、図1の主記憶共有型の並列計算機を使用した、LU分解処理の並列実行方法を示す。図8は本実施例で使用するプログラムの

例であり、図9はそのプログラムを図1の装置で実行する手順を示すフローチャートである。

【0048】以下、本実施例を、図9のフローチャートに基づいて、従来例で示した図7と異なる部分について主に説明する。

【0049】図5に関して説明した通り、LU分解処理での次反復回の処理P (K+M) は、図4に示す行列Aにおいて、点線で囲った領域26に対して処理を実行する。したがって、前反復回の処理Q (K) のうち、図4の行列A内の領域26である第K+M列から第K+2*M-1列の更新処理QS (K) を処理Q (K) から除いた残りの処理 (Q (K) - QS (K)) と処理P (K+M) は並列に実行可能である。処理P (K+M) は処理QS (K) を終了すればいつでも実行可能となる。したがって、図9において、処理Q (K) を並列化する処理63を実行し、処理QS (K) を含む処理Q1' (K) 64、処理Q2' (K) 65ないし処理QP' (K) 66のp個に分割し、そのうち、処理Q1' (K) 64を処理P (K+M) 67と同じプロセッサで、その処理より前に実行することにより、処理Q2' (K) 65ないし処理QP' (K) 66と処理P (K+M) とを並列に実行することができるようになる。

【0050】処理P (K+M) 67をK回目の反復回で実行するために、61、68および69で制御される反

$$K1 = \text{MAX} (K+2M-1, K+M + (N - (K+M) + 1 - \text{Prow}) / p) \quad (1)$$

であらわすことができる。ここで、MAXは最大値を求める関数である。

【0055】さらに、K2ないしK (p-1) は処理Q (K) から処理Q1' (K) 64を除いた処理を、残り

$$Ki = K1 + (i-1) / (p-1) * (N-K1) + 1 \quad (2)$$

であらわすことができる。ここで、KiはK2ないしK (p-1) をあらわす。

【0057】処理の均等分割によるK1、K2、...、K (p-1) の決定処理62は、各反復回ごとに行なえば効率が良いが、対象とする行列は徐々に小さくなるので、数回の反復回ごとに値を更新するようにすることもできる。

【0058】図8に示すプログラムは、プロセッサ台数Pが4の場合である。

【0059】まず、K=1のときの処理P (1) 45を実行し、DO9ループ46で指示された反復処理を実行する。次に前述した式 (式1) および (式2) により、処理を均等に分割するためにK1、K2、K3を決定47する。

【0060】プログラムの並列化記述は、ひとつのプロセッサのみに処理P (K+M) 58を割り付けるので、SECTIONタイプの並列化指示行48ないし53を用いる。各プロセッサの処理は、各SECTION () 文で区切られた範囲を実行する。

【0061】SECTION (1) 49では、処理Q1' (K) 64

を1度だけ実行する必要がある。

【0051】処理Q1' (K) 64、処理Q2' (K) 65ないし処理QP' (K) 66の分割は、処理P (K+M) 67を含めて、全体の処理が均等になるように分割する必要がある。処理Q (K) の分割は、各プロセッサに割り付ける列の数を変更することにより自由に扱うことができる。ここでは、処理Q1' (K) 64で扱う列をK+MからK1とし、処理Q2' (K) 65で扱う列をK1+1からK2とし、処理QP' (K) 66で扱う列をK (p-1) からNとする。

【0052】この処理の均等分割によるK1、K2、...、K (p-1) の決定62の手順を説明する。

【0053】まず、処理P (K+M) に先だって実行する必要のある処理QS (K) は、最低、処理Q1' (K) 64に含まれている必要がある。したがって、K1はK+2*M-1以上となる (条件1)。今仮に、処理P (K+M) 67の演算量を処理Q (K) における列の演算量に換算した時の列数をProwとする。処理Q1' (K) 64と処理P (K+M) 67の演算量の和を全体の1/pにし、かつ条件1を満たすためには、K1の値は次の式、

【0054】

【数1】

のp-1台のプロセッサで均等に分割すれば良い。したがって、K2ないしK (p-1) は次の式、

【0056】

【数2】

(K) 54と処理P (K+M) 58をこの順番に実行する。ここで、処理Q1' (K) 54は行列Aの第K+M列から第K1列までを担当する。ここで、K1は (式1) を用いて求める。(式1) のK1の定義により、処理Q1' (K) 54は前述した処理QS (K) を含むことを保障することができる。

【0062】SECTION (2) 50では、処理Q2' (K) 55を実行する。処理Q2' (K) 55は行列Aの第 (K1+1) 列から第K2列までを担当する。ここで、K2は (式2) を用いて求める。

【0063】SECTION (3) 51では、処理Q3' (K) 56を実行する。処理Q3' (K) 56は行列Aの第 (K2+1) 列から第K3列までを担当する。ここで、K3も (式2) を用いて求める。

【0064】SECTION (4) 52では、処理Q4' (K) 57を実行する。処理Q4' (K) 57は行列Aの第 (K3+1) 列から第N列までを担当する。ここで、K3は (式2) を用いて求める。

【0065】図8に示したプログラムをコンパイルすることにより、図9に示した手順で実行されるオブジェクト

トプログラムが生成される。このコンパイルにより、図8に示す各部分に対応するオブジェクト部分が生成されるとともに、これらのオブジェクト部分の実行を制御するために、並列化処理63と同期処理69を実行するコードがこのオブジェクトプログラムに追加される。このような目的のためのコンパイラはすでに知られている。

【0066】この実施例により、処理Pを実質上、並列処理の中に組み込むことが可能となるために、処理Pの実行時間が実質的に $(1 - 1/p)$ に短縮される。この結果LU分解処理全体の処理時間が短縮される。例えば、図1の形式の並列計算機が4台のプロセッサを有し、かつ、各プロセッサがベクトルプロセッサの場合、行列のサイズNを1000で、処理Pと処理Qの実行時間がほぼ同程度の場合は、本実施例を用いることにより、LU分解処理の実行時間を最大35%程度低減することができる。

【0067】＜実施例2＞本実施例は、図1の主記憶共有型の並列計算機を使用した、LU分解処理の並列実行方法の他の例を提供する。図10は本実施例で使用するプログラムの例であり、図11はこのプログラムを図1の装置で実行させるときの実行手順を示すフローチャートである。

【0068】実施例1においては、図9で示したごとく、各反復ごとに毎回並列化処理63を行なうことにより、処理P(K+M)と次の反復回の処理Q1'(K+M)ないし処理QP'(K+M)との順序関係を保証している。しかし、毎行なう並列化処理63はすべてのプロセッサ間の同期を取るために性能低下の要因となる可能性がある。本実施例では、きめ細かくPost処理やWait処理を用いた同期処理を用いて順序保障処理を行ない、並列化処理を制御変数Kの反復ループの外側に移動させる方法を説明する。ここで用いるPost処理やWait処理は、OSの発行するPost/Waitマクロではなく、コンパイラの並列制御ライブラリのひとつであるPost/Waitライブラリであり、コストが小さい処理である。

【0069】この処理手順を図11を用いて図9と比較して説明する。まず、図9と同様に、処理P(1)101を実行する。次に、制御変数Kにより指示される反復ループの外側に並列化処理102を移動する。このため、処理の均等分割処理103ないし105、反復ループの制御変数Kの更新処理117ないし119および反復ループの判定処理120ないし122は、並列化された後、各プロセッサごとに実行させる。さらに、他のプロセッサの次の反復回の処理との順序関係を保障するために、プロセッサ間でPost処理およびWait処理を実行する必要がある。

【0070】Post処理およびWait処理の設定すべき位置について説明する。基本的な方針はPost処理をできるかぎり早く実行し、Wait処理をできるか

ぎり遅く実行することである。このことにより、各プロセッサの実行の独立性を高め、動的な要因（たとえば、主記憶アクセス競合など）によるWait処理時の不要な同期待ちを軽減することができる。

【0071】まず、処理P(K+M)107を担当するプロセッサの実行順序を説明する。このプロセッサで実行する消去更新処理Q1'(K)を処理QS(K)106と処理(Q1'(K) - QS(K))108の2つに分解する。ここで、処理QS(K)106は、処理P(K+M)107に先だって行なう行列A内の第K+M行から第K+2*M-1行の更新処理である。処理(Q1'(K) - QS(K))108は、このプロセッサが担当する処理Q(K)に関する残りの処理である。処理P(K+M)107を実行することにより、他のプロセッサの次の反復ループの処理Q2'(K+M)109ないし処理QP'(K+M)110を実行することができるので、ここで、Post処理111およびWait処理115ないし116を実行する。ここで、Post処理111は他のp-1台のプロセッサのWait処理と対応することとする。次に、処理(Q1'(K) - Q1S(K))108を実行する。次反復回の処理P(K+2*M)を実行するためには、その前に他のプロセッサの処理Q2'(K)109ないし処理QP'(K)110を実行する必要があるので、ここで、Post処理113ないし114およびWait処理112を実行する。ここで、Wait処理112はp-1台のプロセッサからのPost処理をすべて受けて成立することとする。

【0072】このように、処理Q1'(K)を処理QS(K)106と処理(Q1'(K) - Q1S(K))108の2つに分解することにより、Post処理111は処理P(K+M)107の直後に実行可能となり、処理(Q1'(K) - Q1S(K))108の時間だけpost処理111を他のプロセッサのWait処理115ないし116より早く実行することが可能となる。

【0073】図10に示すプログラムでは、SECTIONタイプの並列化70ないし75は1回だけ行なう。各SECTION()文とSECTION()文の間に各プロセッサで各々実行する制御変数Kの反復ループ76ないし79がある。それぞれの反復ループの中に各演算処理80ないし89がある。複数のPost処理とWait処理を対応させるために、ここでは、イベント変数を引数とする。複数のイベント変数を持つPost処理90は、設定した数のイベントを発行し、Wait処理95ないし97と対応する。複数のイベント変数を持つWait処理94は、設定した数のイベントを対応するPost処理91ないし93から受けることにより成立する。

【0074】図10に示したプログラムをコンパイルすることにより、図11に示した手順で実行されるオブジェクトプログラムが生成される。このことは、実施例1

の場合と同様である。

【0075】＜実施例3＞本実施例は、図2の主記憶分散型の並列計算機を使用した、LU分解処理の並列実行方法を提供する。図13は本実施例で使用するプログラムの例であり、図14はこのプログラムを図2の装置で実行させるときの実行手順を示すフローチャートである。

【0076】このような並列計算機では、図2に示すように、相互結合網5を介して、プロセッサ6ないし8間でのデータの送受信処理が必要となる。このデータの送受信処理を用いて順序保証を行なう。以下では、メッセージ交換型の並列計算機を例に説明する。

【0077】分散記憶型の並列計算機では、まず、データを分割配置する必要がある。ここでは、図12に示すように、行列Aを多段展開の幅Mごとに行を分割して、順次各プロセッサに分割する。このような分割では、各プロセッサが、順番にピボット処理Pを担当することになる。

【0078】この処理手順を図14を用いて説明する。図11と比較して、図14で異なっている部分は、

(1) 同期処理のPost/Wait処理の代わりに、データの送信/受信処理を伴うSend処理およびReceive処理を用いること、(2) 図14の実施例では、つねに一つのプロセッサがピボット処理Pを担当していたのに対して、実施例3では、負荷をできるだけ均等にするために、各プロセッサが交互にピボット処理を担当することである。図14では、2台のプロセッサ(PEaとPEb)で処理を実行する場合を示す。もちろん、3台以上のプロセッサの場合も同様である。処理Q(K)は、プロセッサ数である2つに均等に分割される。それぞれのプロセッサの担当する処理を処理Qa(K)および処理Qb(K)とする。

【0079】このとき、もし、データを送信するSend処理とデータを受信するReceive処理が非同期に実行可能な場合は、Send処理を可能なかぎり先行させて、Receive処理を同様に可能なかぎり遅らせて実行する必要がある。これは、図11のPost/Wait処理を用いた時のスケジューリング方針と同様であり、処理Qa(K)を処理Qs(K)170と処理(Qa(K) - Qs(K))172とに分解することにより、実現可能である。

【0080】まず、それぞれのプロセッサでは、160、161において自プロセッサがどちらのプロセッサであるかを判断する。これは、一般にたとえばUNIXコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。ここでは、それぞれ自プロセッサであると判断した162および163とする。

【0081】次に、プロセッサPEaで、処理P(1)166を行ない、生成した領域L2(図4)のデータをプロセッサPEbに送信するためにSend処理167

を行なう。

【0082】一方、プロセッサPEbは、このデータを受信するReceive処理168を実行する。以下、各プロセッサはそれぞれ制御変数Kの反復ループに入る。

【0083】まず、プロセッサPEbは、処理Qs(K)173を実行し、引き続き、処理P(K+M)174を実行する。これにより、K+M回目の消去処理に用いる領域L2の生成が可能となるために、ただちにプロセッサPEaにSend処理178を実行する。その後、残りの処理(Qb(K) - Qs(K))175を実行し、次反復回であるK+Mステップに手順を進める。

【0084】一方、プロセッサPEaでは、まず処理Qa(K)169で担当するQの領域の更新処理を行ない、その後に、K+Mステップに必要なL2のデータをReceive処理177で受け取る。このとき、プロセッサPEb側のSend処理178は可能なかぎり早く実行されているため、Receive処理177での受信待ちはあまりなく、すぐにデータ受信が可能となる。

【0085】次にK+Mステップに進む。今度は、プロセッサPEa側がピボット処理P(K+2*M)171を担当する。以下の処理は、プロセッサPEbのKステップの手順と同等である。他方のプロセッサPEbでもK+Mステップの処理は、プロセッサPEaのKステップの処理と同様に行なう。

【0086】まず、処理Qs(K+M)170を実行し、引き続き、処理P(K+2*M)171を実行する。これにより、K+2*M回目の消去処理に用いる領域L2の生成が可能となるために、ただちにプロセッサPEbにSend処理179を実行する。その後、残りの処理(Qa(K+M) - Qs(K+M))172を実行し、次反復回であるK+2*Mステップに手順を進めるために、制御変数Kの更新処理181、判定処理183を行なう。

【0087】一方、プロセッサPEbでは、まず処理Qb(K)176で担当するQの領域の更新処理を行ない、その後に、K+2*Mステップに必要なL2のデータをReceive処理180で受け取る。このとき、プロセッサPEa側のSend処理179は可能なかぎり早く実行されているため、Receive処理180での受信待ちはあまりなく、すぐにデータ受信が可能となる。その後、制御変数Kの更新処理182、判定処理184を行なう。

【0088】図13に示すプログラムでは、まず130では自プロセッサがどちらのプロセッサであるかを判断する。これは、一般にたとえばUNIXコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。それぞれのプロセッサの処理は、条件文131ないし133で判別される。図13においては、136以下13

2の直前までがプロセッサPEaの処理で、148以下133の直前までがプロセッサPEbの処理である。

【0089】各プロセッサで交互にピボット処理を担当するために、まず、1回だけ、プロセッサPEaで、処理P(1)136を行ない、その結果をSEND処理145で送信し、プロセッサPEbにおいてRECEIVE処理148で受信する。その後、各プロセッサは、反復ループに入る。各反復ループDO91およびDO92は制御変数Kに関して、2回分の処理を行なっている。そのために、DO91およびDO92ループの間隔の指定が、図13の134、135に示すように、第1、2の実施例に比べて2倍となっている。

【0090】K回目のステップでは、プロセッサPEbにおいて、処理QS(K)141、処理P(K+M)142および処理(Qb(K)-Qs(K))143を行ない、SEND処理149でプロセッサPEaにピボット値を送信する。プロセッサPEaでは、平行して処理Qa(K)137を行ない、RECEIVE処理146でピボット値を受信する。

【0091】K+M回目のステップでは、プロセッサPEaにおいて、処理QS(K+M)138、処理P(K+2*M)139および処理(Qa(K+M)-Qs(K+M))140の演算処理を行ない、SEND処理147でプロセッサPEbにピボット値を送信する。プロセッサPEbでは、平行して処理Qb(K)144を行ない、RECEIVE処理150でピボット値を受信する。

【0092】図13に示したプログラムをコンパイルすることにより、図14に示した手順で実行されるオブジェクトプログラムが生成される。このことは、実施例1の場合と同様である。

【0093】<実施例4>本実施例は、図3に示した、複数のクラスタにより構成された主記憶分散型の並列計算機を使用した、LU分解処理の並列実行方法を提供する。図15は本実施例で使用するプログラムの例であり、図16はこのプログラムを図3の装置で実行させるときの実行手順を示すフローチャートである。

【0094】このような並列計算機では、実施例3のように、データの送受信処理をもとに順序保証を行ない、さらに、各反復回ごとに、クラスタ内のプロセッサ間で第1、第2の実施例のような並列処理を実施する。

【0095】実施例4の実施例3と異なる点は、各ステップ実行内で、クラスタ内のプロセッサ間の並列処理を実行することである。このときの手順は、ここでは図9を用いて説明した実施例1で示した方法を用いる。もちろん、実施例2で示した図11の手順も利用することが可能である。

【0096】本実施例では、2つのクラスタCLaとCLb、および各クラスタ内のプロセッサ数がp台と仮定する。以下、本実施例での処理手順を図16を用いて説

明する。

【0097】図14と比較して、図16で異なる点は、

(1) 図14におけるプロセッサPEaおよびPEbがそれぞれ図16のクラスタCLaおよびCLbに対応すること、(2) 各クラスタ内の各ステップの処理において、クラスタ内のp台のプロセッサにおいて、並列処理が行なわれることの2点である。図16では、2台のクラスタ(CLaとCLb)で処理を実行する場合を示す。もちろん、3台以上のクラスタの場合も同様である。

【0098】このとき、もし、データを送信する送信Send処理とデータを受信するReceive処理が非同期実行可能な場合は、Send処理を可能なかぎり先行させて、Receive処理を同様に可能なかぎり遅らせて実行する必要がある。これは、図14におけるSend処理/Receive処理を用いた時のスケジューリング方針と同様である。

【0099】最初に、クラスタCLaでピボット処理P(1)252を行ない、生成した領域L2(図4参照)のデータをクラスタCLbに送信するためにSend処理253を実行する。

【0100】一方、クラスタCLbはこのデータを受信するReceive処理254を実行する。以下、各クラスタは、それぞれ制御変数Kの反復ループに入る。制御変数Kの制御は、279ないし282で行なう。処理Q(K)は、クラスタ数である2つに均等に分割される。それぞれのクラスタの担当する処理を処理Qa(K)および処理Qb(K)とする。

【0101】まず、クラスタCLbは、K回目の反復処理において、クラスタ内のp台のプロセッサを用いて並列処理256を行なう。このときの並列処理の分割の方法は、実施例1から実施例3にしてしたのと同じの方法で可能である。

【0102】ピボット処理P(K+M)263を担当するプロセッサでは、処理QS(K)262を実行し、引き続き、ピボット処理P(K+M)263を実行する。これにより、K+M回目の消去処理に用いる領域L2の生成が可能となるために、ただちにクラスタCLaにSend処理268を実行する。その後、残りの処理(Qb(K)-Qs(K))264を実行する。クラスタCLb内の他のプロセッサは、並列にそれぞれ処理Qb2(K)265および処理Qbp(K)266を実行する。Kステップ内でクラスタ内のプロセッサ間の同期は、並列化処理256で行なわれる。その後、K+Mステップに手順を進める。

【0103】一方、クラスタCLaでは、まず処理Qa1(K)259で担当するQの領域の更新処理を行ない、その後、K+Mステップに必要なL2のデータをReceive処理267で受け取る。このとき、クラスタCLb側のSend処理268は可能なかぎり早く

実行されているため、Receive処理267での受信待ちはあまりなく、すぐにデータ受信が可能となる。クラスタCLa内の他のプロセッサは、並列にそれぞれ処理Qa2(K)260および処理Qap(K)261を実行する。Kステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理255で行なわれる。その後、K+Mステップに手順を進める。

【0104】次にK+Mステップに進む。今度は、クラスタCLa側がピボット処理P(K+2*M)270を担当する。以下の処理は、クラスタCLbのKステップの手順と同等である。

【0105】処理Qs(K+M)269を実行し、引き続き、ピボット処理P(K+2*M)270を実行する。これにより、K+2*M回目の消去処理に用いる領域L2の生成が可能となるために、ただちにクラスタCLbにSend処理277を実行する。その後、残りの処理(Qa1(K+M)-Qs(K+M))271を実行する。クラスタCLa内の他のプロセッサは、並列にそれぞれ処理Qa2(K)272および処理Qap(K)273を実行する。K+Mステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理257で行なわれる。その後、K+2*Mステップに手順を進める。

【0106】一方、クラスタCLbでは、まず処理Qb1(K+M)274において、担当する領域の更新処理を行ない、その後、K+2*Mステップに必要なL2のデータをReceive処理278で受け取る。このとき、クラスタCLa側のSend処理277は可能なかぎり早く実行されているため、Receive処理278での受信待ちはあまりなく、すぐにデータ受信が可能となる。クラスタCLb内の他のプロセッサは、並列にそれぞれ処理Qb2(K)275および処理Qbp(K)276を実行する。K+Mステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理258で行なわれる。その後、K+2*Mステップに手順を進める。

【0107】図15に示すプログラムでは、2台のクラスタ(CL aとCL b)でかつ各クラスタ内のプロセッサ数がそれぞれ2台の場合を示す。もちろん、3台以上のクラスタの場合、あるいは、クラスタ内のプロセッサ台数を3台以上にすることに可能である。図15中、206以下202の直前までがクラスタCLaの処理で、208以下203の直前までがクラスタCLbの処理である。ここでは、まず200では自クラスタがどちらのクラスタであるかを判断する。これは、一般にたとえばUNIXコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。

【0108】各クラスタで交互にピボット処理を行なうために、DO91ループ204およびDO92ループ205は制御変数Kに関して、2回分の処理を行なっている。そのために、実施例3と同様に、DO91およびDO92ループの間隔の指定が、実施例1、2に比べて2

倍となっている。

【0109】各クラスタ内では、各反復ステップKおよびK+Mごとに、並列処理指示文209ないし224により、並列化の指示を行なう。

【0110】図15の例では、クラスタCLaにおいては、ステップKに対しては、209から212の間の並列指示行により並列処理が規定されており、演算処理225および226、RECEIVE処理237を行なう。ステップK+Mでは、213から216の間の並列指示行により並列処理が規定されており、演算処理227ないし229および230、SEND処理239を行なう。

【0111】クラスタCLbにおいては、ステップKに対しても、同様に217から220の間の並列指示行により並列処理が規定されており、演算処理231ないし234、SEND処理238を行なう。ステップK+Mでは、221から224の間の並列指示行により並列処理が規定されており、演算処理235および236、RECEIVE処理240を行なう。

【0112】なお、第3および実施例4では、この処理では、データの送受信のためのSend処理やReceive処理を他の演算処理と並列に実行することが必要となる。もちろん、このSend処理やReceive処理が専用の通信プロセッサで動作可能である場合は、それらの通信に要する処理時間を除いたのちに、演算を担当するプロセッサ内で演算量の処理を均等にする必要がある。

【0113】図15に示したプログラムをコンパイルすることにより、図16に示した処理手順が実行されるオブジェクトプログラムが生成されることは、実施例1の場合と同様である。

【0114】

【発明の効果】本来は逐次処理されるべき第1の処理とその後に行なわれるべき、並列実行可能な第2の処理とを有し、第2の処理の完了を待って再度第1、第2の処理を実行するように、第1、第2の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、プロセッサが待ち状態にある時間を軽減することが出来、計算機資源の有効な利用を図ることが出来る。それにより、この反復処理全体の処理時間を軽減することが可能になる。

【図面の簡単な説明】

【図1】従来の主記憶共有型並列計算機の概略構造を示す図。

【図2】従来の主記憶分散型並列計算機の概略構造を示す図。

【図3】従来のクラスタ結合型並列計算機の概略構造を示す図。

【図4】従来の多段同時消去LU分解法の概要を示す図。

【図 5】従来の多段同時消去 LU 分解法のプログラム例を示す図。

【図 6】従来の反復計算処理のフローチャート。

【図 7】並列化された反復計算処理のフローチャート。

【図 8】本発明の実施例 1 で使用するプログラム例を示す図。

【図 9】実施例 1 による反復計算処理のフローチャート。

【図 10】実施例 2 で使用するプログラム例を示す図。

【図 11】本発明の実施例 2 による反復計算処理のフローチャート。

【図 12】分散記憶型計算機向けのデータの分割を示す図。

【図 13】本発明の実施例 3 で使用するプログラム例を示す図。

【図 14】実施例 3 による反復計算処理のフローチャート。

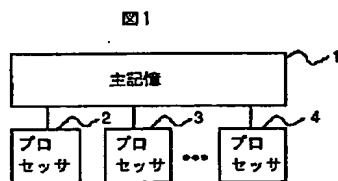
【図 15】本発明の実施例 4 で使用するプログラム例を示す図。

【図 16】実施例 4 による反復計算処理のフローチャート。

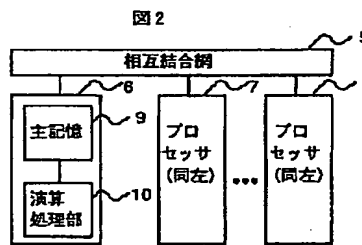
【符号の説明】

20、34、42…ループ制御、27、31、36…逐次的処理 P (K)、28、32…並列化可能処理 Q (K)、29…並列指示、38 ないし 40…並列化可能処理 Q (K) を分割した処理

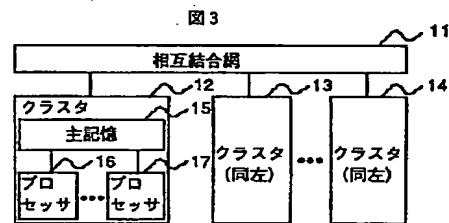
【図 1】



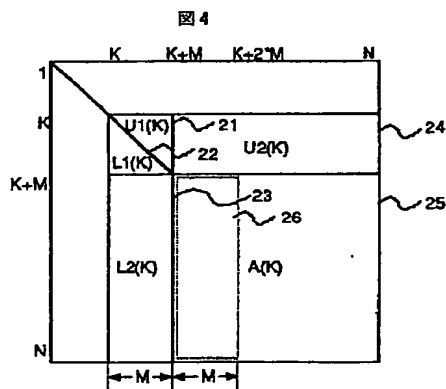
【図 2】



【図 3】



【図 4】

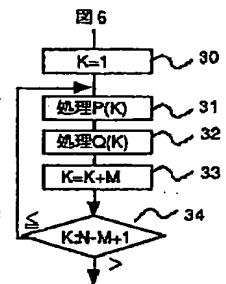


【図 5】

```

DO 9 K=1, N-M+1, M
DO 1 KM=K, K+M-1
  ピボット処理(KM列中の最大値検索と行交換)
  L1(K), U1(K), L2(K)領域の消去更新
1  CONTINUE
* POPTION PARALLEL
DO 2 J=K+M, N
  U2(K)およびA(K)領域に対する行交換
  U1(K), L1(K)領域を用いたU2(K)領域の消去更新
  U2(K), L2(K)領域を用いたA(K)領域の消去更新
2  CONTINUE
9  CONTINUE
  
```

【図 6】

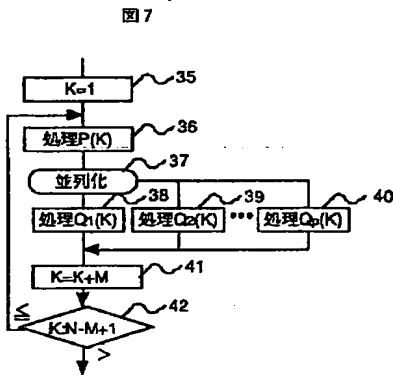


【図 8】

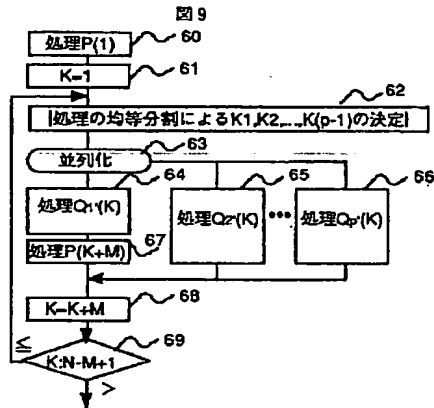
```

図 8
処理 P(1)
DO 9 K=1, N-M+1, M
[処理の均等分割による K1, K2, K3 の決定]
* POPTION PARALLEL_SECTIONS
* POPTION SECTION (1)
  処理 Q1' (K) [J = K+M, K1 を担当]
  処理 P(K+M)
* POPTION SECTION (2)
  処理 Q2' (K) [J = K1+1, K12 を担当]
* POPTION SECTION (3)
  処理 Q3' (K) [J = K2+1, K3 を担当]
* POPTION SECTION (4)
  処理 Q4' (K) [J = K3+1, N を担当]
* POPTION END PARALLEL_SECTIONS
9 CONTINUE
  
```

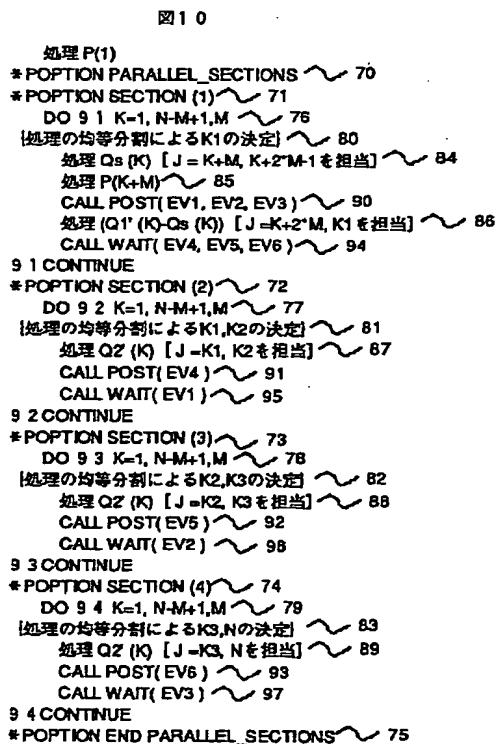
【図7】



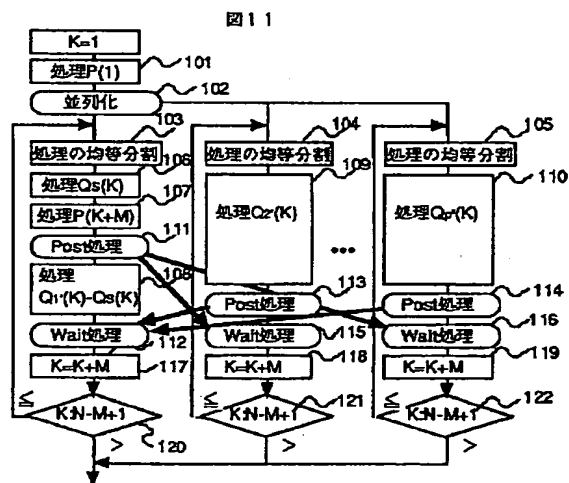
【図9】



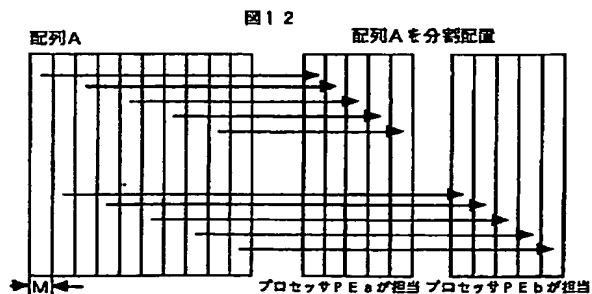
【図10】



【図11】



【図12】



【図 13】

図 13

【自 P E 番号 MYPEN の設定】 130
 IF (MYPEN.EQ.PEa) THEN 131
 処理 P(1) 136
 CALL SEND(L2(1)) 145
 DO 9 1 K=1, N-2*M+1, 2*M 134

C Step K
 処理 Qa (K) 137
 CALL RECEIVE(L2(K+M)) 148

C Step K+M
 処理 Qs (K+M) 138
 処理 P(K+2*M) 139
 CALL SEND(L2(K+2*M)) 147
 処理 (Qa (K)-Qs (K)) 140

9 1 CONTINUE
 ELSE IF (MYPEN.EQ.PEb) THEN 132
 CALL RECEIVE(L2(1)) 148
 DO 9 2 K=1, N-2*M+1, 2*M 135

C Step K
 処理 Qs (K) 141
 処理 P(K+M) 142
 CALL SEND(L2(K+M)) 149
 処理 (Qb (K)-Qs (K)) 143

C Step K+M
 処理 Qb (K+M) 144
 CALL RECEIVE(L2(K+2*M)) 150

9 2 CONTINUE
 END IF 133

【図 15】

図 15

【自 クラスタ番号 MYCLN の設定】 200
 IF (MYCLN.EQ.CLa) THEN 205
 処理 P(1) 205
 CALL SEND(L2(1)) 204
 DO 9 1 K=1, N-2*M+1, 2*M 204

C Step K
 *POPTION PARALLEL_SECTIONS 209
 *POPTION SECTION (1) 210
 処理 Qa1 (K) 225
 CALL RECEIVE(L2(K+M)) 237
 *POPTION SECTION (2) 211
 処理 Qa2 (K) 226
 *POPTION END PARALLEL_SECTIONS 212

C Step K+M
 *POPTION PARALLEL_SECTIONS 213
 *POPTION SECTION (1) 214
 処理 Qs (K+M) 227
 処理 P(K+2*M) 228
 CALL SEND(L2(K+2*M)) 239
 処理 (Qa1 (K+M)-Qs (K+M)) 229
 *POPTION SECTION (2) 215
 処理 Qa2 (K+M) 230
 *POPTION END PARALLEL_SECTIONS 216

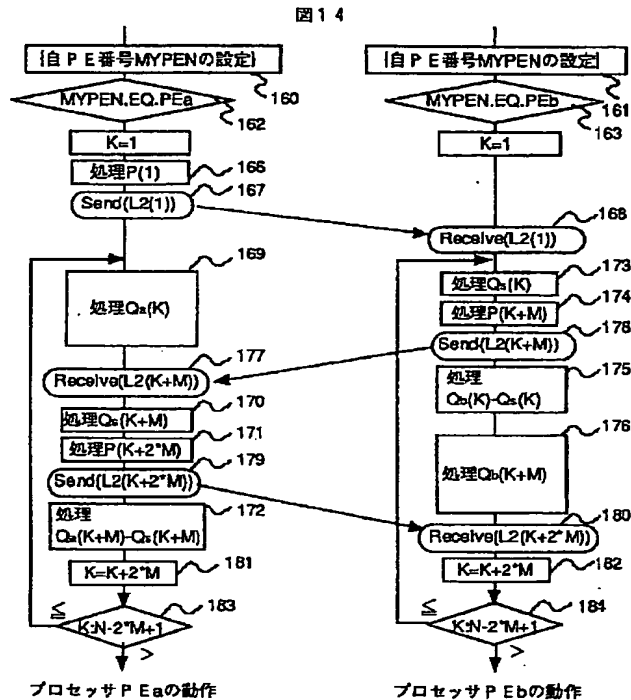
9 1 CONTINUE
 ELSE IF (MYPEN.EQ.CLb) THEN 202
 CALL RECEIVE(L2(1)) 208
 DO 9 2 K=1, N-2*M+1, 2*M 205

C Step K
 *POPTION PARALLEL_SECTIONS 217
 *POPTION SECTION (1) 218
 処理 Qs (K) 231
 処理 P(K+M) 232
 CALL SEND(L2(K+M)) 238
 処理 (Qb1 (K)-Qs (K)) 233
 *POPTION SECTION (2) 219
 処理 Qb2 (K) 234
 *POPTION END PARALLEL_SECTIONS 220

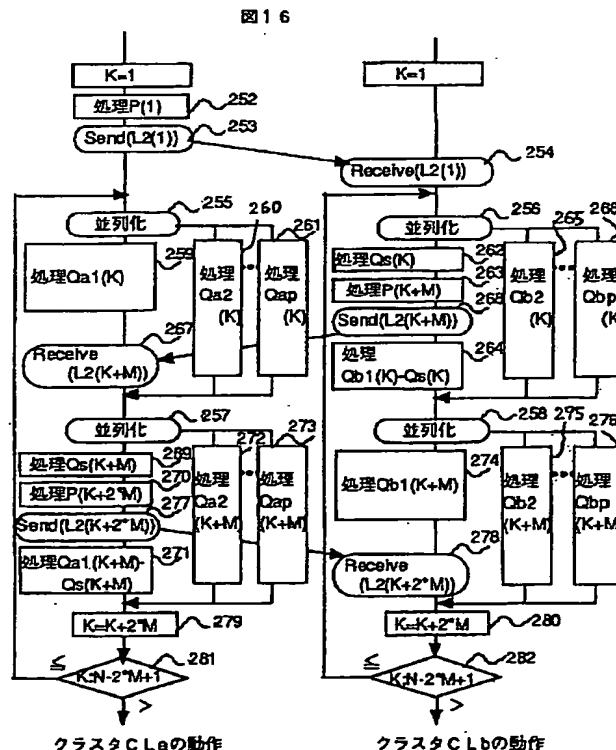
C Step K+M
 *POPTION PARALLEL_SECTIONS 221
 *POPTION SECTION (1) 222
 処理 Qb1 (K+M) 235
 CALL RECEIVE(L2(K+2*M)) 240
 *POPTION SECTION (2) 223
 処理 Qb2 (K+M) 236
 *POPTION END PARALLEL_SECTIONS 224

9 2 CONTINUE
 END IF 203

【図 14】



【図 16】



フロントページの続き

(72) 発明者 玉置 由子
東京都国分寺市東恋ヶ窪 1 丁目 280 番地
株式会社日立製作所中央研究所内
(72) 発明者 榑原 忠幸
東京都国分寺市東恋ヶ窪 1 丁目 280 番地
株式会社日立製作所中央研究所内

(72) 発明者 朝家 真知子
東京都国分寺市東恋ヶ窪 1 丁目 280 番地
株式会社日立製作所中央研究所内
(72) 発明者 保田 淑子
東京都国分寺市東恋ヶ窪 1 丁目 280 番地
株式会社日立製作所中央研究所内